Towards Formal Specification and Verification of Transactional Memory: Machine-Checked Proofs

Victor Luchangco (joint work with Simon Doherty and Mark Moir)
Sun Labs
Scalable Synchronization Research Group
Transactional Memory

- Facilitate concurrent programming
- Transactions encapsulate abstract operations
  - operations appear to take effect atomically

As a foundation used by other concurrent programs, we want to be especially sure that TM implementations are correct!
Vision

- Precise formal specifications for transactional memory
  - one size will not fit all uses
- Precise formal models of TM algorithms
- Rigorous proofs that algorithms meet specifications
Our approach

• Model algorithms as automata
  • relatively straightforward
  • well-developed theory: invariants, etc.

• Model specifications as automata
  • well-developed theory: simulation relations (refinements), etc.
  • hierarchical proofs: intermediate specifications

• Machine-checked proofs
  • greater confidence; repeatable
  • reusable: for different algorithms or specifications
Hierarchical, reusable proofs

- High-level specification captures abstract requirements
- Intermediate specification for implementation approach
- Model algorithms at multiple levels
High-level specification

- extOrder: external order (among transactions)
- ops_t: operations of transaction t
- invokedCommit_t: has t invoked commit
  - whether t is “visible”
- status_t/pendingOp_t: bookkeeping
  - <active>, commitPending, committed, aborted

- validCommit
- validFail
- validResp
“Write-latest” specification

- mem: sequence of memory states
- beginIdx_t: “timestamp” of state at beginning of txn t
- rdSet_t: read set of t
- wrSet_t: write set of t
- pc_t: bookkeeping

Covers implementation approach of many TM systems
Machine-checked proofs (using PVS)

- Formalize automata in PVS language
  - need to formalize basic tools: automata, sequences
- State lemmas (also in PVS language)
- Prove lemma using PVS prover
  - theorem prover (not model checker)
    - equational rewriting-based, with decision procedures
Current status

• Formalized high-level and “write-latest” specifications
  • and automata, simulations (and sequences, etc.)
• Formalized “coarse-grain” version of TL2
• Proved simulation relation from write-latest spec to high-level spec
  • mostly invariants and lemmas about write-latest spec
  • lots of “helper” lemmas about sequences
  • still need to prove having a simulation relation implies implementation
    • infinite traces make this a bit tricky
Next steps

- Formalize abstract version of NOrec
- Prove that abstract NOrec implements TMS2
- Prove that coarse-grain TL2 implements TMS2
- Prove that fine-grain TL2 implements coarse-grain TL2
- Reformulate automata, traces, etc.
  - build library of useful general lemmas
- Consider other TM algorithms